9

Detektion einfacher Kurven

In Kap. 7 haben wir gezeigt, wie man mithilfe von geeigneten Filtern Kanten finden kann, indem man an jeder Bildposition die Kantenstärke und möglicherweise auch die Orientierung der Kante bestimmt. Der darauf folgende Schritt bestand in der Entscheidung (z. B. durch Anwendung einer Schwellwertoperation auf die Kantenstärke), ob an einer Bildposition ein Kantenpunkt vorliegt oder nicht, mit einem binären Kantenbild (*edge map*) als Ergebnis. Das ist eine sehr frühe Festlegung, denn natürlich kann aus der beschränkten („myopischen“) Sicht eines Kantenfilters nicht zuverlässig ermittelt werden, ob sich ein Punkt tatsächlich auf einer Kante befindet oder nicht. Man muss daher davon ausgehen, dass in dem auf diese Weise produzierten Kantenbild viele vermeintliche Kantenpunkte markiert sind, die in Wirklichkeit zu keiner echten Kante gehören, und andererseits echte Kantenpunkte fehlen. Kantenbilder enthalten daher in der Regel zahlreiche irrelevante Strukturen und gleichzeitig sind wichtige Strukturen häufig unvollständig. Das Thema dieses Kapitels ist es, in einem vorläufigen, binären Kantenbild auffällige und möglicherweise bedeutsame Strukturen aufgrund ihrer Form zu finden.

9.1 Auffällige Strukturen

Ein intuitiver Ansatz zum Auffinden größerer Bildstrukturen könnte darin bestehen, beginnend bei einem beliebigen Kantenpunkt benachbarte Kantenpixel schrittweise aneinander zu fügen und damit die Konturen von Objekten zu bestimmen. Das könnte man sowohl im kontinuierlichen Kantenbild (mit Kantenstärke und Orientierung) als auch im binären *edge map* versuchen. In beiden Fällen ist aufgrund von Unterbrechungen, Verzweigungen und ähnlichen Mehrdeutigkeiten mit Problemen zu rechnen und ohne zusätzliche Kriterien und Informationen über

die Art der gesuchten Objekte bestehen nur geringe Aussichten auf Erfolg. Das lokale, sequentielle Verfolgen von Konturen (*contour tracing*) ist daher ein interessantes Optimierungsproblem [49] (s. auch Kap. 11.2).

Eine völlig andere Idee ist die Suche nach global auffälligen Strukturen, die von vornherein gewissen Formeigenschaften entsprechen. Wie das Beispiel in Abb. 9.1 zeigt, sind für das menschliche Auge derar-

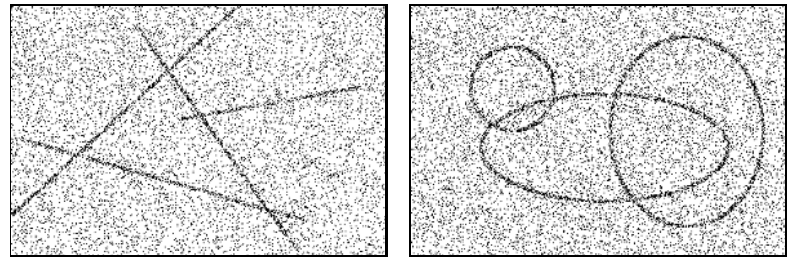


Abbildung 9.1
Das menschliche Sehsystem findet auffällige Bildstrukturen spontan auch unter schwierigen Bedingungen.

tige Strukturen auch dann auffällig, wenn keine zusammenhängenden Konturen gegeben sind, Überkreuzungen vorliegen und viele zusätzliche Elemente das Bild beeinträchtigen. Es ist auch heute weitgehend unbekannt, welche Mechanismen im biologischen Sehen für dieses spontane Zusammenfügen und Erkennen unter derartigen Bedingungen verantwortlich sind. Eine Technik, die zumindest eine vage Vorstellung davon gibt, wie derartige Aufgabenstellungen mit dem Computer möglicherweise zu lösen sind, ist die so genannte „Hough-Transformation“, die wir nachfolgend näher betrachten.

9.2 Hough-Transformation

Die Methode von Paul Hough – ursprünglich als US-Patent [38] publiziert und oft als „Hough-Transformation“ (HT) bezeichnet – ist ein allgemeiner Ansatz, um beliebige, parametrisierbare Formen in Punktverteilungen zu lokalisieren [21, 42]. Zum Beispiel können viele geometrische Formen wie Geraden, Kreise und Ellipsen mit einigen wenigen Parametern beschrieben werden. Da sich gerade diese Formen besonders häufig im Zusammenhang mit künstlichen, von Menschenhand geschaffenen Objekten finden, sind sie für die Analyse von Bildern besonders interessant (Abb. 9.2).

Betrachten wir zunächst den Einsatz der HT zur Detektion von Geraden in binären Kantenbildern, eine relativ häufige Anwendung. Eine Gerade in 2D kann bekanntlich mit zwei reellwertigen Parametern beschrieben werden, z. B. in der klassischen Form

$$y = kx + d, \tag{9.1}$$

Abbildung 9.2
Einfache geometrische Formen, wie gerade, kreisförmige oder elliptische Segmente, erscheinen häufig im Zusammenhang mit künstlichen bzw. technischen Objekten.

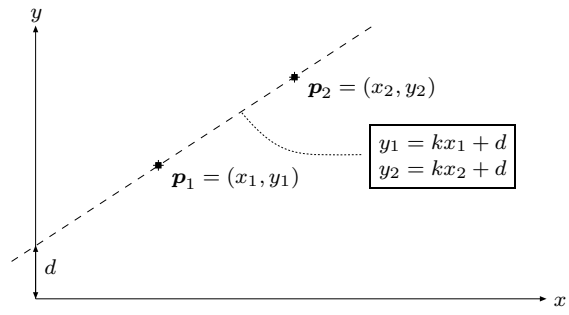


Abbildung 9.3
Zwei Bildpunkte $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ liegen auf derselben Geraden, wenn $y_1 = kx_1 + d$ und $y_2 = kx_2 + d$ für ein bestimmtes k und d.

wobei k die Steigung und d die Höhe des Schnittpunkts mit der y-Achse bezeichnet (Abb. 9.3). Eine Gerade, die durch zwei gegebene (Kanten-)Punkte $\boldsymbol{p}_1 = (x_1, y_1)$ und $\boldsymbol{p}_2 = (x_2, y_2)$ läuft, muss daher folgende beiden Gleichungen erfüllen:

$$y_1 = kx_1 + d \quad \text{und} \quad y_2 = kx_2 + d \tag{9.2}$$

für $k, d \in \mathbb{R}$. Das Ziel ist nun, jene Geradenparameter k und d zu finden, auf denen möglichst *viele* Kantenpunkte liegen, bzw. jene Geraden, die möglichst viele dieser Punkte „erklären“. Wie kann man aber feststellen, wie viele Punkte auf einer bestimmten Geraden liegen? Eine Möglichkeit wäre etwa, alle möglichen Geraden in das Bild zu „zeichnen“ und die Bildpunkte zu zählen, die jeweils exakt auf einer bestimmten Geraden liegen. Das ist zwar grundsätzlich möglich, wegen der großen Zahl an Geraden aber nicht besonders effizient.

9.2.1 Parameterraum

Die Hough-Transformation geht an dieses Problem auf dem umgekehrten Weg heran, indem sie alle möglichen Geraden ermittelt, die durch einen

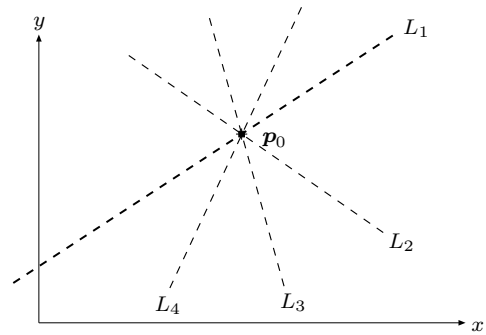


Abbildung 9.4
Geradenbündel durch einen Bildpunkt. Für alle möglichen Geraden L_j durch den Punkt $\boldsymbol{p}_0 = (x_0, y_0)$ gilt $y_0 = k_j x_0 + d_j$ für geeignete Parameter k_j, d_j.

einzelnen, gegebenen Bildpunkt laufen. Jede Gerade L_j, die durch einen Punkt $\boldsymbol{p}_0 = (x_0, y_0)$ läuft, muss die Gleichung

$$L_j : y_0 = k_j x_0 + d_j \tag{9.3}$$

für geeignete Werte von k_j, d_j erfüllen. Die Menge der Lösungen für k_j, d_j in Gl. 9.3 entspricht einem Bündel von unendlich vielen Geraden, die alle durch den gegebenen Punkt $\boldsymbol{p}_0$ laufen (Abb. 9.4). Für ein bestimmtes k_j ergibt sich die zugehörige Lösung für d_j aus Gl. 9.3 als

$$d_j = -x_0 k_j + y_0 \,, \tag{9.4}$$

also wiederum eine lineare Funktion (Gerade), wobei nun k_j, d_j die *Variablen* und x_0, y_0 die (konstanten) *Parameter* der Funktion sind. Die Lösungsmenge $\{(k_j, d_j)\}$ von Gl. 9.4 beschreibt die Parameter aller möglichen Geraden L_j, die durch einen Bildpunkt $\boldsymbol{p}_0 = (x_0, y_0)$ führen.

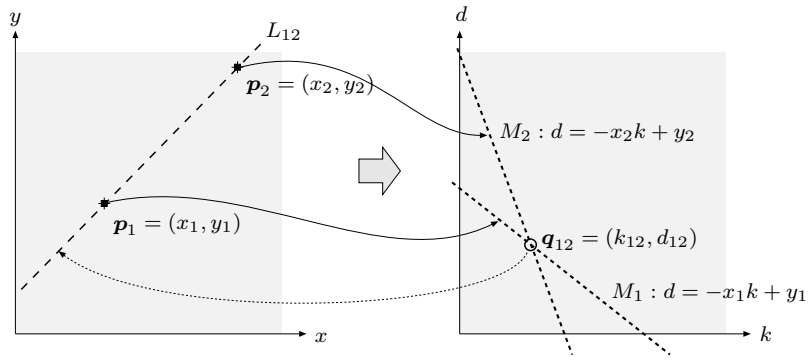
Für einen *beliebigen* Bildpunkt $\boldsymbol{p}_i = (x_i, y_i)$ entspricht Gl. 9.4 einer Geraden

$$M_i : d = -x_i k + y_i \tag{9.5}$$

mit den Parametern $-x_i, y_i$ im so genannten *Parameterraum* (auch „Hough-Raum“ genannt), der durch die Koordinaten k, d aufgespannt wird. Der Zusammenhang zwischen dem Bildraum und dem Parameterraum lässt sich folgendermaßen zusammenfassen:

Bildraum (x, y)		*Parameterraum* (k, d)	
Punkt	$\boldsymbol{p}_i = (x_i, y_i)$	$M_i : d = -x_i k + y_i$	Gerade
Gerade	$L_j : y = k_j x + d_j$	$\boldsymbol{q}_j = (k_j, d_j)$	Punkt

Jedem Punkt $\boldsymbol{p}_i$ und seinem zugehörigen Geradenbüschel im Bildraum entspricht also exakt eine Gerade M_i im Parameterraum. Am meisten sind wir jedoch an jenen Stellen interessiert, an denen sich Geraden im Parameterraum *schneiden*. Wie am Beispiel in Abb. 9.5 gezeigt, schneiden sich die Geraden M_1 und M_2 an der Position $\boldsymbol{q}_{12} = (k_{12}, d_{12})$ im Parameterraum, die den Parametern jener Geraden im Bildraum entspricht, die sowohl durch den Punkt $\boldsymbol{p}_1$ als auch durch den Punkt $\boldsymbol{p}_2$



(a) x/y-Bildraum (b) k/d-Parameterraum

Abbildung 9.5
Zusammenhang zwischen Bildraum und Parameterraum. Die Parameterwerte für alle möglichen Geraden durch den Bildpunkt $\boldsymbol{p}_i = (x_i, y_i)$ im Bildraum (a) liegen im Parameterraum (b) auf einer Geraden M_i. Umgekehrt entspricht jeder Punkt $\boldsymbol{q}_j = (k_j, d_j)$ im Parameterraum einer Geraden L_j im Bildraum. Der Schnittpunkt der zwei Geraden M_1, M_2 an der Stelle $\boldsymbol{q}_{12} = (k_{12}, d_{12})$ im Parameterraum zeigt an, dass im Bildraum eine Gerade L_{12} mit zwei Punkten und den Parametern k_{12} und d_{12} existiert.

verläuft. Je mehr Geraden M_i sich an einem Punkt im Parameterraum schneiden, umso mehr Bildpunkte liegen daher auf der entsprechenden Geraden im Bildraum! Allgemein ausgedrückt heißt das:

> Wenn sich N Geraden an einer Position (k', d') im *Parameterraum* schneiden, dann liegen auf der entsprechenden Geraden $y = k'x + d'$ im *Bildraum* insgesamt N Bildpunkte.

9.2.2 Akkumulator-Array

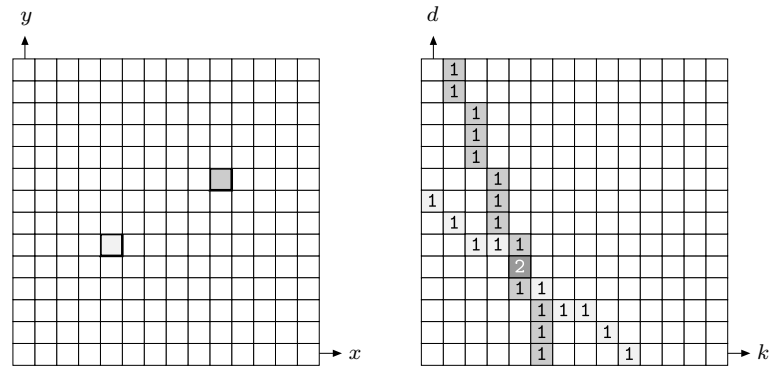
Das Ziel, die dominanten Bildgeraden zu finden, ist daher gleichbedeutend mit dem Auffinden jener Koordinaten im Parameterraum, an denen sich viele Parametergeraden schneiden. Genau das ist die Intention der HT. Um die HT zu berechnen, benötigen wir zunächst eine diskrete Darstellung des kontinuierlichen Parameterraums mit entsprechender Schrittweite für die Koordinaten k und d. Um die Anzahl der Überschneidungen im Parameterraum zu berechnen, wird jede Parametergerade M_i additiv in dieses „Akkumulator-Array“ gezeichnet, indem jede durchlaufene Zelle um den Wert 1 erhöht wird (Abb. 9.6).

9.2.3 Eine bessere Geradenparametrisierung

Leider ist die Geradenrepräsentation in Gl. 9.1 in der Praxis nicht wirklich brauchbar, denn es gilt $k = \infty$ für vertikale Geraden. Eine bessere Lösung ist die so genannte *Hesse'sche Normalform* (HNF) der Geradengleichung

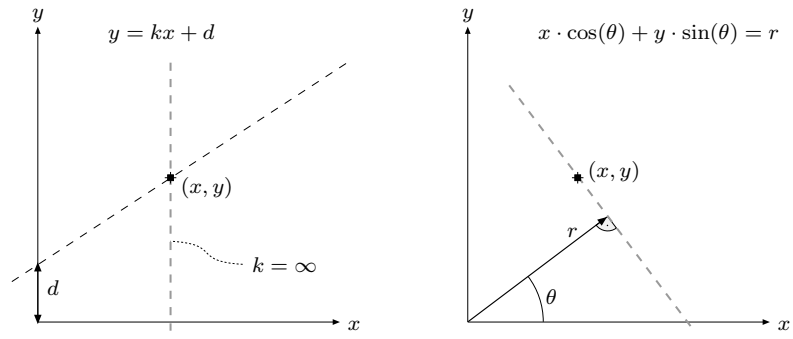
$$x \cdot \cos(\theta) + y \cdot \sin(\theta) = r\,, \tag{9.6}$$

die keine derartigen Singularitäten aufweist und außerdem eine lineare Quantisierung ihrer Parameter, des Winkels θ und des Radius r, ermöglicht (s. Abb. 9.7). Mit der HNF-Parametrisierung hat der Parameterraum die Koordinaten θ, r und jedem Bildpunkt $\boldsymbol{p}_i = (x_i, y_i)$ entspricht darin die Relation



(a) Bildraum (b) Akkumulator-Array

Abbildung 9.6
Kernidee der Hough-Transformation. Das Akkumulator-Array ist eine diskrete Repräsentation des Parameterraums (k, d). Für jeden gefundenen Bildpunkt (a) wird eine diskrete Gerade in den Parameterraum (b) gezeichnet. Diese Operation erfolgt *additiv*, d. h., jede durchlaufene Array-Zelle wird um den Wert 1 erhöht. Der Wert jeder Zelle des Akkumulator-Arrays entspricht der Anzahl von Parametergeraden, die sich dort schneiden (in diesem Fall 2).



(a) (b)

Abbildung 9.7
Parametrisierung von Geraden in 2D. In der üblichen k, d-Parametrisierung (a) ergibt sich bei vertikalen Geraden ein Problem, weil in diesem Fall $k = \infty$. Die Hesse'sche Normalform (b), bei der die Gerade durch den Winkel θ und den Abstand vom Ursprung r dargestellt wird, vermeidet dieses Problem.

$$r_{x_i,y_i}(\theta) = x_i \cdot \cos(\theta) + y_i \cdot \sin(\theta)\,, \tag{9.7}$$

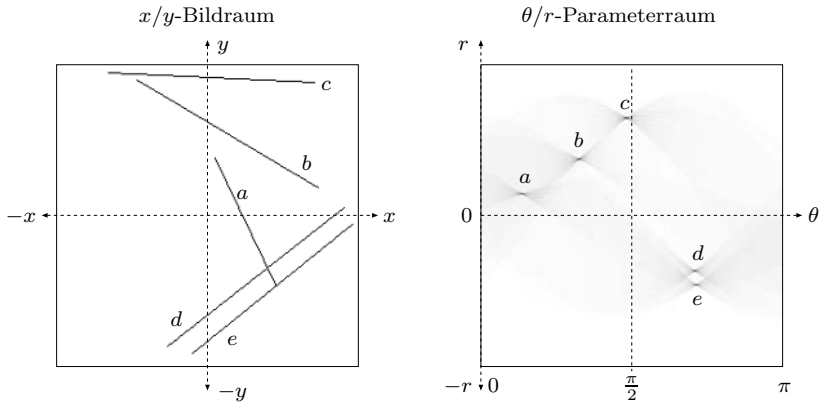
für den Winkelbereich $0 \le \theta < \pi$ (s. Abb. 9.8). Wenn wir das Zentrum des Bilds als Referenzpunkt für die x/y-Bildkoordinaten benutzen, dann ist der mögliche Bereich für den Radius auf die Hälfte der Bilddiagonale beschränkt, d. h.

$$-r_{\max} \le r_{x,y}(\theta) \le r_{\max}, \quad \text{wobei} \quad r_{\max} = \tfrac{1}{2}\sqrt{M^2 + N^2} \tag{9.8}$$

für ein Bild der Breite M und der Höhe N.

9.3 Implementierung der Hough-Transformation

Der grundlegende Hough-Algorithmus für die Geradenparametrisierung mit der HNF (Gl. 9.6) ist in Alg. 9.1 gezeigt. Ausgehend von einem binären Eingangsbild $I(u, v)$, das bereits markierte Kantenpixel (Wert 1) enthält, wird im ersten Schritt ein zweidimensionales Akkumulator-Array erzeugt und gefüllt. Im zweiten Schritt (FindMaxLines()) wird,



(a) (b)

Abbildung 9.8
Bildraum und Parameterraum für die HNF-Parametrisierung.

```
 1: HoughLines(I)
 2:     Set up a two-dimensional array Acc[θ, r] of counters, initialize to 0
 3:     Let (u_c, v_c) be the center coordinates of the image I
 4:     for all image coordinates (u, v) do
 5:         if I(u, v) is an edge point then
 6:             (x, y) ← (u−u_c, v−v_c)          ▷ relative coordinate to center
 7:             for θ_i = 0 ... π do
 8:                 r_i = x cos(θ_i) + y sin(θ_i)
 9:                 Increment Acc[θ_i, r_i]
10:     MaxLines ← FindMaxLines(Acc, K)
11:         ▷ return the list of parameter pairs (θ_j, r_j) for K strongest lines
12:     return MaxLines.
```

Algorithmus 9.1
Einfacher Hough-Algorithmus für Geraden. Dieser liefert eine Liste der Parameter (θ, r) für die K stärksten Geraden im binären Kantenbild $I(u, v)$.

wie nachfolgend beschrieben, das Akkumulator-Array nach maximalen Einträgen durchsucht und ein Vektor von Parameterwerten für die K stärksten Geraden

$$MaxLines = \big((\theta_1, r_1), (\theta_2, r_2), \ldots (\theta_K, r_K)\big)$$

wird zurückgegeben.

9.3.1 Füllen des Akkumulator-Arrays

Eine direkte Implementierung des ersten Teils von Alg. 9.1 zeigt Prog. 9.1, bestehend aus einer einzigen Java-Klasse `LinearHT`. Das Akkumulator-Array (`houghArray`) ist als zweidimensionales `int`-Array definiert. Aus dem ursprünglichen Bild `imp` wird die HT durch Erzeugen einer neuen Instanz der Klasse `LinearHT` berechnet, d. h.

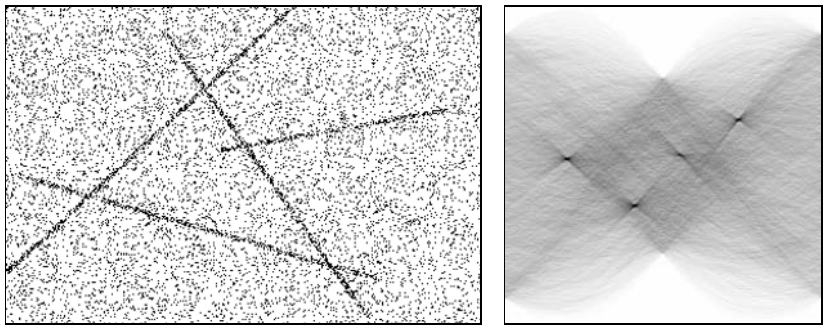
```
LinearHT HT = new LinearHT(imp, 256, 256);
```

Die letzten beiden Parameter (256, 256) spezifizieren die Anzahl der diskreten Schritte für den Winkel θ und den Radius r. `imp` wird als

Programm 9.1
Hough-Transformation für Geraden (Java-Implementierung).

```
1  class LinearHT {
2    ImageProcessor ip;  // reference to original image
3    int nAng, nRad;     // number of steps for angle and radius
4    double dAng, dRad;  // stepsize of angle and radius
5    int xCtr, yCtr;     // x/y-coordinate of image center
6    int[][] houghArray; // Hough accumulator
7
8    //constructor method:
9    LinearHT(ImageProcessor ip, int aSteps, int rSteps) {
10     this.ip = ip;
11     xCtr = ip.getWidth()/2; yCtr = ip.getHeight()/2;
12     nAng = aSteps; dAng = (Math.PI/nAng);
13     nRad = rSteps;
14     double rMax = Math.sqrt(xCtr*xCtr + yCtr*yCtr);
15     dRad = (2*rMax)/nRad;
16     houghArray = new int[nAng][nRad];
17     fillHoughAccumulator();
18   }
19
20   void fillHoughAccumulator() {
21     for (int v = 0; v < ip.getHeight(); v++) {
22       for (int u = 0; u < ip.getWidth(); u++) {
23         if (ip.getPixel(u, v) > 0) {
24           doPixel(u, v);
25         }
26       }
27     }
28   }
29
30   void doPixel(int u, int v) {
31     int x = u-xCtr, y = v-yCtr;
32     for (int a = 0; a < nAng; a++) {
33       double theta = dAng * a;
34       int r = (int) Math.round(
35         (x*Math.cos(theta) + y*Math.sin(theta)) / dRad) + nRad
               /2;
36       if (r >= 0 && r < nRad) {
37         houghArray[a][r]++;
38       }
39     }
40   }
41 }
```

`ByteProcessor` (d. h. als 8-Bit-Grauwertbild) angenommen und jeder Bildwert größer als null wird als Kantenpixel interpretiert. Die Anwendung dieses Programms auf ein stark verrauschtes Kantenbild ist in Abb. 9.9 gezeigt.



(a) (b)

Abbildung 9.9
Hough-Transformation für Geraden. Das Originalbild (a) hat eine Größe von 360 × 240 Pixel, womit sich ein maximaler Radius (Abstand vom Bildzentrum) $r_{\max} \approx 216$ ergibt. Im zugehörigen Parameterraum (b) wird eine Rasterung von jeweils 256 Schritten sowohl für den Winkel $\theta = 0 \ldots \pi$ (horizontale Achse) als auch für den Radius $r = -r_{\max} \ldots r_{\max}$ (vertikale Achse) verwendet. Die vier dunklen Maximalwerte im Akkumulator-Array entsprechen den Parametern der vier Geraden im Originalbild. Die Intensität wurde zur besseren Sichtbarkeit invertiert.

9.3.2 Auswertung des Akkumulator-Arrays

Der nächste Schritt ist die Lokalisierung der Maximalwerte im Akkumulator-Array $Acc[\theta, r]$. Wie in Abb. 9.9 (b) deutlich zu erkennen ist, schneiden sich auch beim Vorliegen von geometrisch exakten Bildgeraden die zugehörigen Akkumulator-Kurven nicht *genau* in einzelnen Zellen, sondern die Schnittpunkte sind über eine gewisse Umgebung verteilt. Die Ursache sind Rundungsfehler aufgrund der diskreten Koordinatengitter. Durch dieses Problem wird die Auswertung des Akkumulator-Arrays zum schwierigsten Teil der HT, und es gibt dafür auch keine Patentlösung. Zu den einfachen Möglichkeiten gehören folgende zwei Ansätze (s. Abb. 9.10):

Variante A: Schwelloperation

Zunächst unterziehen wir das Akkumulator-Array einer Schwellwertoperation mit dem Wert t_a und setzen dabei alle Akkumulator-Werte $Acc[\theta, r] < t_a$ auf null. Die verbleibenden Regionen (Abb. 9.10 (b)) könnte man z. B. mit einer morphologischen *Closing*-Operation (s. Abschn. 10.3.2) auf einfache Weise bereinigen. Als Nächstes lokalisieren wir die noch übrigen Regionen in $Acc[\theta, r]$ (z. B. mit einer der Techniken in Abschn. 11.1), berechnen die Schwerpunkte der Regionen (s. Abschn. 11.4.3) und verwenden deren (nicht ganzzahlige) Koordinaten als Parameter der gefundenen Geraden. Weiters ist die Summe der Akkumulator-Werte innerhalb einer Region ein guter Indikator für die Stärke (Anzahl der Bildpunkte) der Geraden.

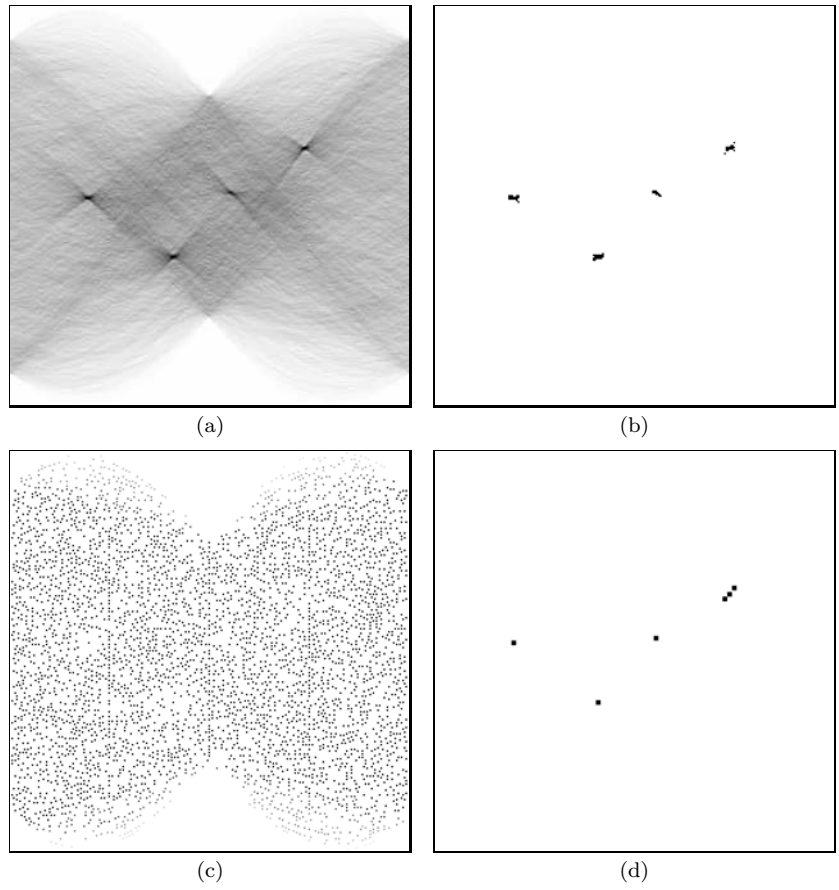


Abbildung 9.10
Bestimmung lokaler Maximalwerte im Akkumulator-Array. Ursprüngliche Verteilung der Werte im Hough-Akkumulator (a). **Variante A:** *Schwellwertoperation* mit 50% des Maximalwerts (b) – die verbleibenden Regionen entsprechen den vier dominanten Bildgeraden. Die Koordinaten der Schwerpunkte dieser Regionen ergeben eine gute Schätzung der echten Geradenparameter. **Variante B:** Durch *Non-Maximum Suppression* entsteht zunächst eine große Zahl lokaler Maxima (c), die durch eine anschließende Schwellwertoperation reduziert wird (d).

Variante B: Non-Maximum Suppression

Die Idee dieser Methode besteht im Auffinden lokaler Maxima im Akkumulator-Array durch Unterdrückung aller *nicht* maximalen Werte.[1] Dazu wird für jede Zelle in $Acc[\theta, r]$ festgestellt, ob ihr Wert höher ist als die Werte aller ihrer Nachbarzellen. Ist dies der Fall, dann wird der bestehende Wert beibehalten, ansonsten wird die Zelle auf null gesetzt (Abb. 9.10 (c)). Die (ganzzahligen) Koordinaten der verbleibenden Spitzen sind potentielle Geradenparameter und deren jeweilige Höhe entspricht der Stärke der Bildgeraden. Diese Methode kann natürlich mit einer Schwellwertoperation verbunden werden, um die Anzahl der Kandidatenpunkte einzuschränken. Das entsprechende Ergebnis zeigt Abb. 9.10 (d).

9.3.3 Erweiterungen der Hough-Transformation

Was wir bisher gesehen haben, ist nur die einfachste Form der Hough-Transformation. Für den praktischen Einsatz sind unzählige Verbesse-

[1] Non-Maximum Suppression wird auch in Abschn. 8.2.3 zur Isolierung von Eckpunkten verwendet.

rungen und Verfeinerungen möglich und oft auch unumgänglich. Hier eine kurze und keineswegs vollständige Liste von Möglichkeiten.

Bias

Da der Wert eine Zelle im Hough-Akkumulator der Anzahl der Bildpunkte auf der entsprechenden Geraden entspricht, können lange Geraden grundsätzlich höhere Werte als kurze Geraden erzielen. Zum Beispiel kann eine Gerade in der Nähe einer Bildecke nie dieselbe Anzahl von Treffern in ihrer Akkumulator-Zelle erreichen wie eine Gerade entlang der Bilddiagonalen (Abb. 9.11). Wenn wir daher im Ergebnis nur

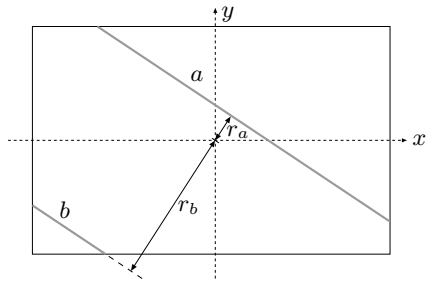


Abbildung 9.11
Bias-Problem. Innerhalb der endlichen Bildfläche sind Geraden mit einem kleinen Abstand r vom Zentrum i. Allg. länger als Geraden mit großem r. Zum Beispiel ist die maximal mögliche Zahl von Akkumulator-Treffern für Gerade a wesentlich höher als für Gerade b.

nach den maximalen Einträgen suchen, ist die Wahrscheinlichkeit hoch, dass kürzere Geraden überhaupt nicht gefunden werden. Eine Möglichkeit, diesen systematischen Fehler zu kompensieren, besteht darin, jeden Akkumulator-Eintrag $Acc[\theta, r]$ bezüglich der maximal möglichen Zahl von Bildpunkten $MaxHits[\theta, r]$ auf der Geraden mit den Parametern θ, r zu normalisieren:

$$Acc'[\theta, r] \leftarrow \frac{Acc[\theta, r]}{MaxHits[\theta, r]} \quad \text{für} \quad MaxHits[\theta, r] > 0. \tag{9.9}$$

$MaxHits[\theta, r]$ kann z. B. durch Berechnung der Hough-Transformation auf ein Bild mit den gleichen Dimensionen ermittelt werden, in dem alle Pixel aktiviert sind, oder durch ein zufälliges Bild, in dem die Pixel gleichförmig verteilt sind.

Endpunkte von Bildgeraden

Die einfache Version der Hough-Transformation liefert zwar die Parameter der Bildgeraden, nicht aber deren Endpunkte. Das nachträgliche Auffinden der Endpunkte bei gegebenen Geradenparametern ist nicht nur aufwendig, sondern reagiert auch empfindlich auf Diskretisierungs- bzw. Rundungsfehler. Eine Möglichkeit ist, die Koordinaten der Extrempunkte einer Geraden bereits innerhalb der Berechnung des Akkumulator-Arrays zu berücksichtigen. Dazu wird jede Akkumulator-Zelle durch zwei

zusätzliche Koordinatenpaare $(start_X, start_Y)$, (end_X, end_Y) ergänzt, d. h.

$$Acc[\theta, r] = (count, start_X, start_Y, end_X, end_Y).$$

Die Koordinaten für die beiden Endpunkte jeder Geraden werden beim Füllen des Akkumulator-Arrays mitgezogen, sodass sie am Ende des Vorgangs jeweils den am weitesten auseinander liegenden Endpunkten der Geraden entsprechen. Natürlich muss bei der Auswertung des Akkumulator-Arrays darauf geachtet werden, dass beim eventuellen Zusammenfügen von Akkumulator-Zellen in diesem Fall auch die Koordinaten der Endpunkte entsprechend berücksichtigt werden müssen.

Berücksichtigung von Kantenstärke und -orientierung

Die Ausgangsdaten für die Hough-Transformation ist üblicherweise ein Kantenbild, das wir bisher als binäres 0/1-Bild mit potentiellen Kantenpunkten betrachtet haben. Das ursprüngliche Ergebnis einer Kantendetektion enthält jedoch zusätzliche Informationen, die für die HT verwendet werden können, insbesondere die Kantenstärke $E(u, v)$ und die lokale Kantenrichtung $\Phi(u, v)$ (s. Abschn. 7.3).

Die *Kantenstärke* $E(u, v)$ ist besonders einfach zu berücksichtigen: Anstatt eine getroffene Akkumulator-Zelle nur um 1 zu erhöhen, wird der Wert der jeweiligen Kantenstärke addiert, d. h.

$$Acc[\theta, r] \leftarrow Acc[\theta, r] + E(u, v).$$

Mit anderen Worten, starke Kantenpunkte tragen auch mehr zum akkumulierten Zellwert bei als schwächere.

Die lokale *Kantenorientierung* $\Phi(u, v)$ ist ebenfalls hilfreich, denn sie schränkt den Bereich der möglichen Orientierungswinkel einer Geraden im Bildpunkt (u, v) ein. Die Anzahl der zu berechnenden Akkumulator-Zellen entlang der θ-Achse kann daher, abhängig von $\Phi(u, v)$, auf einen Teilbereich reduziert werden. Dadurch wird nicht nur die Effizienz des Verfahrens verbessert, sondern durch die Reduktion von irrelevanten „votes“ im Akkumulator auch die Trennschärfe der HT insgesamt erhöht (s. beispielsweise [47, S. 483]).

Hierarchische Hough-Transformation

Die Genauigkeit der Ergebnisse wächst mit der Größe des Parameterraums. Eine Größe von 256 entlang der θ-Achse bedeutet z. B. eine Schrittweite der Geradenrichtung von $\frac{\pi}{256} \approx 0.7°$. Eine Vergrößerung des Akkumulators führt zu feineren Ergebnissen, bedeutet aber auch zusätzliche Rechenzeit und insbesondere einen höheren Speicherbedarf. Die Idee der hierarchischen HT ist, schrittweise wie mit einem „Zoom“ den Parameterraum gezielt zu verfeinern. Zunächst werden mit einem relativ grob aufgelösten Parameterraum die wichtigsten Geraden gesucht,

dann wird der Parameterraum um die Ergebnisse herum mit höherer Auflösung „expandiert“ und die HT rekursiv wiederholt. Auf diese Weise kann trotz eines beschränkten Parameterraums eine relativ genaue Bestimmung der Parameter erreicht werden.

9.4 Hough-Transformation für Kreise und Ellipsen

9.4.1 Kreise und Kreisbögen

Geraden in 2D haben zwei Freiheitsgrade und sind daher mit zwei reellwertigen Parametern vollständig spezifiziert. Ein Kreis in 2D benötigt *drei* Parameter, z. B. in der Form

$$Circle = (\bar{x}, \bar{y}, \rho),$$

wobei $\bar{x}$, $\bar{y}$ die Koordinaten des Mittelpunkts und ρ den Kreisradius bezeichnen (Abb. 9.12). Ein Punkt $\boldsymbol{p} = (u, v)$ liegt auf einem Kreis, wenn

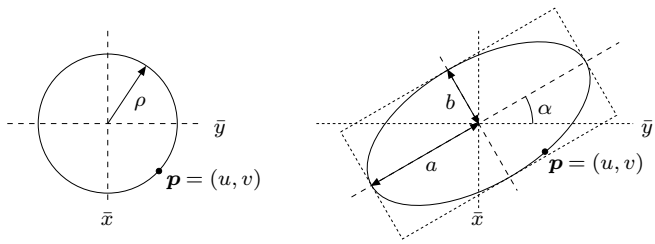


Abbildung 9.12
Parametrisierung von Kreisen und Ellipsen in 2D.

die Bedingung

$$(u - \bar{x})^2 + (v - \bar{y})^2 = \rho^2 \tag{9.10}$$

gilt. Wir benötigen daher für die Hough-Transformation einen dreidimensionalen Parameterraum $Acc[\bar{x}, \bar{y}, \rho]$, um Kreise (und Kreisbögen) mit beliebiger Position und Radius in einem Bild zu finden. Im Unterschied zur HT für Geraden besteht allerdings keine einfache, funktionale Abhängigkeit der Koordinaten im Parameterraum – wie kann man also jene Parameterkombinationen $(\bar{x}, \bar{y}, \rho)$ finden, die Gl. 9.10 für einen bestimmten Bildpunkt $\boldsymbol{p} = (u, v)$ erfüllen? Ein „brute force“-Ansatz wäre, schrittweise und exhaustiv alle Zellen des Parameterraums auf Gültigkeit der Relation in Gl. 9.10 zu testen, wie in Alg. 9.2 beschrieben.

Eine bessere Idee gibt uns Abb. 9.13, aus der wir sehen, dass die Koordinaten der passenden Mittelpunkte im Hough-Raum selbst wiederum Kreise bilden. Wir müssen daher für einen Bildpunkt $\boldsymbol{p} = (u, v)$ nicht den gesamten, dreidimensionalen Parameterraum durchsuchen, sondern brauchen nur in jeder ρ-Ebene des Akkumulator-Arrays die Zellen entlang eines entsprechenden Kreises zu erhöhen. Dazu lässt sich jeder Standardalgorithmus zum Generieren von Kreisen verwenden, z. B. eine Variante des bekannten *Bresenham*-Algorithmus [10].

Algorithmus 9.2
Exhaustiver Hough-Algorithmus zum Auffinden von Kreisen.

```
1:  HoughCircles(I)
2:      Set up a three-dimensional array Acc[x̄, ȳ, ρ] and initialize to 0
3:      for all image coordinates (u, v) do
4:          if I(u, v) is an edge point then
5:              for all (x̄_i, ȳ_i, ρ_i) in the accumulator space do
6:                  if (u − x̄_i)² + (v − ȳ_i)² = ρ_i² then
7:                      Increment Acc[x̄_i, ȳ_i, ρ_i]
8:      MaxCircles ← FindMaxCircles(Acc)   ▷ a list of tuples (x̄_j, ȳ_j, ρ_j)
9:      return MaxCircles.
```

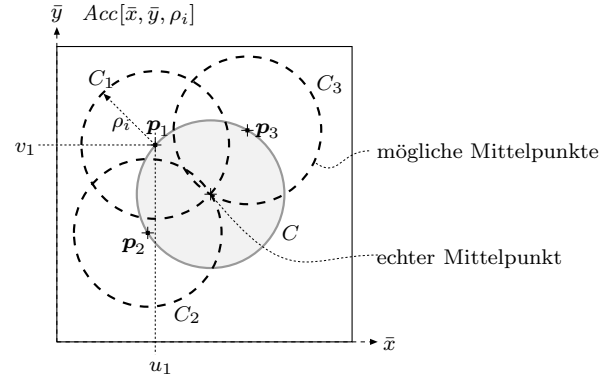


Abbildung 9.13
Hough-Transformation für Kreise. Die Abbildung zeigt eine Ebene des dreidimensionalen Akkumulator-Arrays $Acc[\bar{x}, \bar{y}, \rho]$ für einen bestimmten Kreisradius $\rho = \rho_i$. Die Mittelpunkte aller Kreise, die durch einen gegebenen Bildpunkt $\boldsymbol{p}_1 = (u_1, v_1)$ laufen, liegen selbst wieder auf einem Kreis C_1 (- - -) mit dem Radius ρ_i und dem Mittelpunkt $\boldsymbol{p}_1$. Analog dazu liegen die Mittelpunkte der Kreise, die durch $\boldsymbol{p}_2$ und $\boldsymbol{p}_3$ laufen, auf den Kreisen C_2 bzw. C_3. Die Zellen entlang der drei Kreise C_1, C_2, C_3 mit dem Radius ρ_i werden daher im Akkumulator-Array durchlaufen und erhöht. Die Kreise haben einen gemeinsamen Schnittpunkt im echten Mittelpunkt des Bildkreises C, wo drei „Treffer“ im Akkumulator-Array zu finden sind.

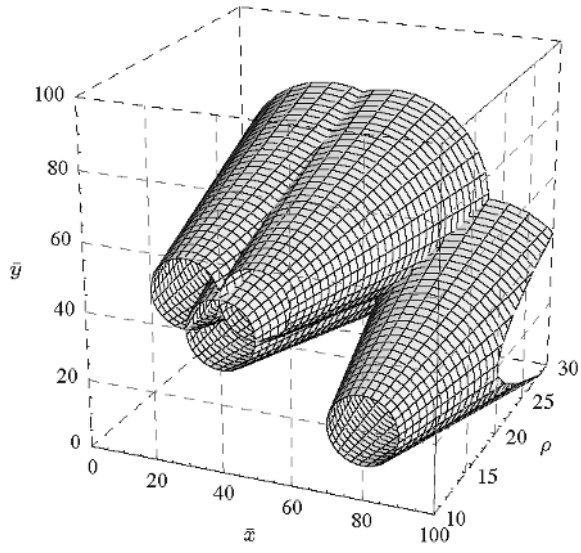
Abb. 9.14 zeigt die räumliche Struktur des dreidimensionalen Parameterraums für Kreise. Für einen Bildpunkt $\boldsymbol{p}_k = (u_k, v_k)$ wird in jeder Ebene entlang der ρ-Achse (für $\rho_i = \rho_{\min} \dots \rho_{\max}$) ein Kreis mit dem Mittelpunkt (u_k, v_k) und dem Radius ρ_i durchlaufen, d. h. entlang einer kegelförmigen Fläche. Die Parameter der dominanten Kreise findet man wiederum als Koordinaten der Akkumulator-Zellen mit den meisten „Treffern“, wo sich also die meisten Kegelflächen schneiden, wobei das *Bias*-Problem (s. Abschn. 9.3.3) natürlich auch hier zutrifft. Kreisbögen werden in gleicher Weise gefunden, allerdings ist die Anzahl der möglichen Treffer proportional zur Bogenlänge.

9.4.2 Ellipsen

In einer perspektivischen Abbildung erscheinen Kreise in der dreidimensionalen Realität in 2D-Abbildungen meist als Ellipsen, außer sie liegen auf der optischen Achse und werden frontal betrachtet. Exakt kreisförmige Strukturen sind daher in gewöhnlichen Fotografien relativ selten anzutreffen. Die Hough-Transformation funktioniert natürlich auch für Ellipsen, allerdings ist die Methode wegen des größeren Parameterraums wesentlich aufwendiger.

Eine allgemeine Ellipse in 2D hat 5 Freiheitsgrade und benötigt zu ihrer Beschreibung daher auch 5 Parameter, z. B.

$$Ellipse = (\bar{x}, \bar{y}, r_a, r_b, \alpha),$$



3D-Parameterraum:
$\bar{x}, \bar{y} = 0 \ldots 100$
$\rho = 10 \ldots 30$

Bildpunkte $\boldsymbol{p}_k$:
$\boldsymbol{p}_1 = (30, 50)$
$\boldsymbol{p}_2 = (50, 50)$
$\boldsymbol{p}_3 = (40, 40)$
$\boldsymbol{p}_4 = (80, 20)$

Abbildung 9.14
3D-Parameterraum für Kreise. Für jeden Bildpunkt $\boldsymbol{p}_k = (u_k, v_k)$ werden die Zellen entlang einer kegelförmigen Fläche im dreidimensionalen Akkumulator-Array $Acc[\bar{x}, \bar{y}, \rho]$ durchlaufen (inkrementiert).

wobei $(\bar{x}, \bar{y})$ die Koordinaten des Mittelpunkts, (r_a, r_b) die beiden Radien und α die Orientierung der Hauptachse bezeichnen (Abb. 9.12). Um Ellipsen in beliebiger Größe, Lage und Orientierung mit der Hough-Transformation zu finden, würde man daher einen 5-dimensionalen Parameterraum mit geeigneter Auflösung in jeder Dimension benötigen. Eine einfache Rechnung zeigt allerdings den enormen Aufwand: Bei einer Auflösung von nur $128 = 2^7$ Schritten in jeder Dimension ergeben sich bereits 2^{35} Akkumulator-Zellen, was bei einer Implementierung als 4-Byte-*Integers* einem Speicherbedarf von 2^{37} Bytes bzw. 128 Gigabytes entspricht. Auch der für das Füllen und für die Auswertung dieses riesigen Parameterraums notwendige Rechenaufwand lässt die Methode wenig praktikabel erscheinen.

Eine interessante Alternative ist in diesem Fall die *verallgemeinerte Hough-Transformation*, mit der grundsätzlich beliebige zweidimensionale Formen detektiert werden können [4,42]. Die Form der gesuchten Kontur wird dazu punktweise in einer Tabelle kodiert und der zugehörige Parameterraum bezieht sich auf die Position (x_c, y_c), den Maßstab S und die Orientierung θ der Form. Er ist damit höchstens vierdimensional, also kleiner als jener der Standardmethode für allgemeine Ellipsen.

9.5 Aufgaben

Aufg. 9.1. Implementieren Sie die Hough-Transformation zum Auffinden von Geraden unter Berücksichtigung der Endpunkte, wie in Abschn. 9.3.3 beschrieben.

Aufg. 9.2. Realisieren Sie eine *hierarchische* Form der Hough-Transformation (S. 166) für Geraden zur genauen Bestimmung der Parameter.

Aufg. 9.3. Implementieren Sie die Hough-Transformation zum Auffinden von Kreisen und Kreissegmenten mit variablem Radius. Verwenden Sie dazu einen schnellen Algorithmus zum Generieren von Kreisen im Akkumulator-Array, wie in Abschn. 9.4 beschrieben.